TELNYX

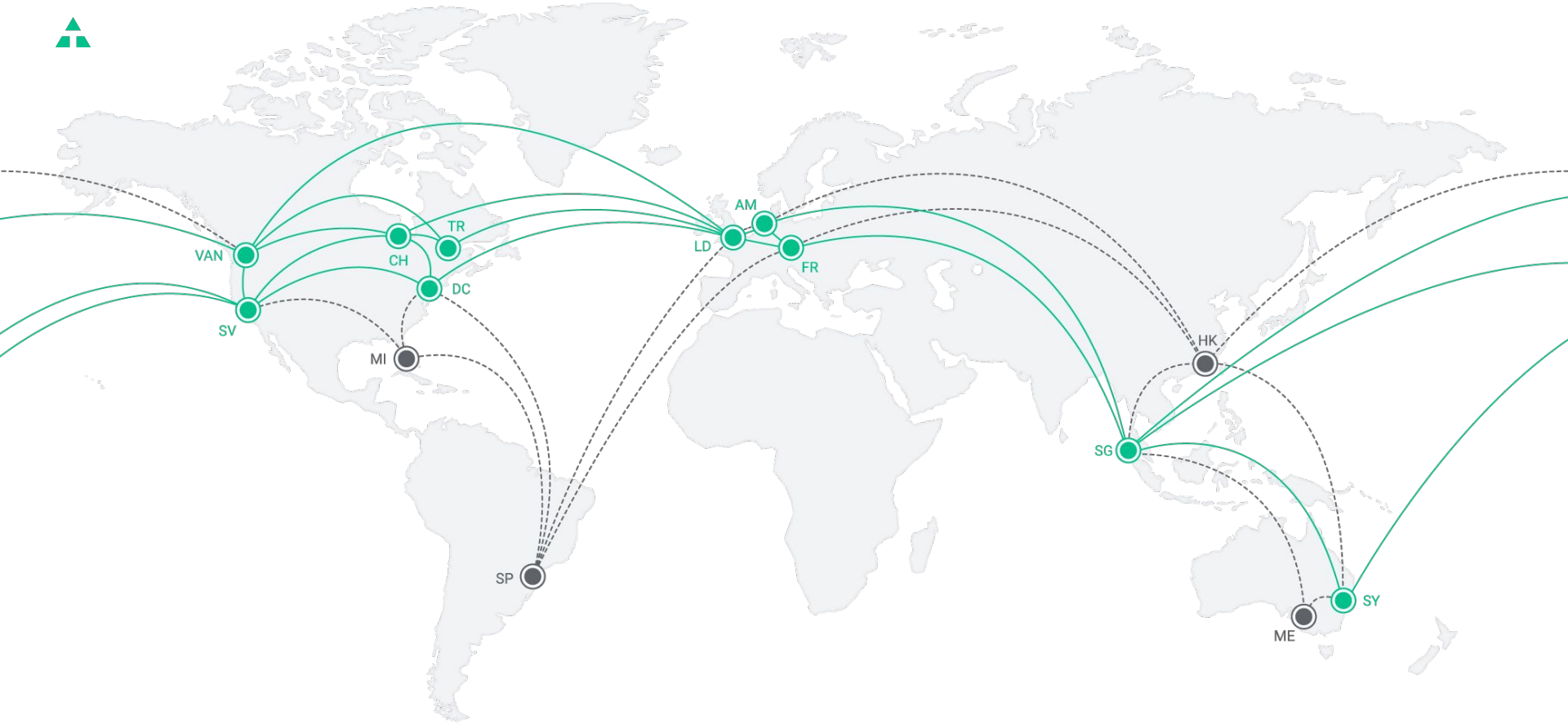
Building Observable Elixir Services

Pawel Szafran

June 2019

Hey, I'm Pawel

- Building backend systems for over 10 years: Java, Clojure and Scala
- For many years worked for large Swiss investment banks: market risk, equity clearing and settlement, equity derivatives, digital archiving
- Summer 2018: impressed by Elixir and OTP, read a book, played with some code ;)
- Joined Telnyx in Jan 2019 as Remote Elixir Engineer



POINT OF PRESENCE

PLANNED POINT OF PRESENCE

FIBER BACKBONE

PLANNED FIBER BACKBONE

Call Control

TELNYX WEBHOOKS	APP COMMANDS
Incoming call	
	Answer call
Call Answered	
	Play Audio
Playback Started	
Playback Ended	
	Gather DTMF
DTMF digit: 1	

Why observable?

- When you run Elixir services (microservices) at scale, how can you tell how well they are performing?
- Stop developing in the dark and understand what's going on with your services
- Know that something is wrong before your clients tell you
- Know service health and uptime
- Understand traffic patterns, success rates, latency profiles, performance bottlenecks
- Optimize MTTD and MTTR
- Save on your debugging time
- Make decisions based on data!

“Talk is cheap. Show me the code.”

Linus Torvalds

... less slides, more code, more demos

- **Principles vs. Implementation**

  Make your services observable.

  How you do it is far less important.

Services in the dark

DEMO

- N = 2
- 2 services, 2 instances each
- Running on local k8s

Test PROD

- Make your tests behave like your most valuable type of client
- Run critical business scenarios very, very often... like all the time
- You can run less critical business scenarios in a separate pack less often
- Run tests from outside of your normal clusters/datacenters/networks
- Alert!

Collect logs

- Logs should be easy to search and explore
- Developers should not be told to “log less” - antipattern
- Logs should not be very expensive to operate
- In most cases: better to have solid logs for last 48h, than unusable logs for last 7d
- **DEMO**
  - Log tailing: Stern
  - Logger
  - Logging backend: Loki

Propagate request ID

- `x-request-id` request/response header
- Do you want to allow you clients to send it?
- Also propagate it over gRPC, NATS, etc.
- **DEMO**
  - Phoenix
  - Logger
  - Tesla
  - Loki
  - across process boundary

Collect service API metrics

- Request throughput
- Success rates
- Latency: percentiles please!
- Apdex: measure user satisfaction
- Measure by operation
- Alert!
- **DEMO**
  - Fortio
  - Prometheus
  - Grafana
  - Dashboards as source code

Collect BEAM metrics

- Collect from all instances in the same place
- Diagnose single instance using `:observer_cli` and `:recon`
- **DEMO**
  - Prometheus + Grafana

Collect service call metrics

- Throughput
- Success rates
- Latency
- Measure by operation
- **DEMO**
  - Prometheus
  - Grafana
  - Tesla
  - `:telemetry`

Collect traces – distributed tracing

- Trace request path as it travels across a complex system
- Open standards:
  - Backend agnostic
  - OpenTracing and OpenCensus... are merging!
- Attach request ID or replace it with trace ID?
- **DEMO**
  - OpenCensus
  - Jaeger
  - Logger
  - OpenCensus can do metrics as well

Consider service mesh

- Istio
  - discovery
  - load balancing
  - failure recovery
  - **metrics and monitoring**
  - A/B testing
  - canary rollouts
  - rate limiting
  - access control
  - authentication

Q&A

Put the Telnyx API to work.
developers.telnyx.com
portal.telnyx.com

Use promo code **ELIXIR19** for a
$50 (190 zł) credit.